Mini Project Report on
Fault Detection on Process Stations

submitted by

Fathimath Shahana.A(FIT21AE017)

in fulfillment of the requirements for the award of the degree of

Bachelor of Technology

Department of Electronics And Instrumentation
Federal Institute of Science And Technology (FISAT)
Angamaly – 683577

July 9, 2024

Department of Electronics And Instrumentation
Federal Institute of Science And Technology (FISAT)
Angamaly - 683577
MAY 2024

Focus on Excellence

CERTIFICATE

This is to certify that this report **FAULT DETECTION IN PROCESS STATIONS** is a bonafide report of the work carried out as part oF the mini project, during the B.Tech Program by **Fathimath Shahana.A** towards the partial fulfillment of the requirements for the award of degree of **Bachelor of Technology**, under our guidance and supervision and that this work has not been submitted elsewhere for the award of any degree.

Project Guide Dept.of EIE FISAT	**Project Coordinator** Dept.of EIE FISAT

Head of the Dept
Dept.of EIE
FISAT

Acknowledgments

we would like to thank the almighty for all the blessings then We would like to express our deepest gratitude to all those who have supported and guided us throughout the completion of this mini project. We would like to extend our sincere thanks to **Dr. Abi P Mathew**, the Head of the Department of Electronics and Instrumentation, Federal Institute of Science and Technology(FISAT) for providing us with the necessary resources to carry out our project. We are also thankful to our project guide and coordinator, **Mr.Anil Johny** and **Ms. Sreevidhya P**, for their invaluable guidance, expertise, and continuous support throughout the duration of our project and also their valuable insights, suggestions, and feedback that greatly contributed to the successful completion of our mini project. Furthermore, we would like to express our gratitude to our friends, classmates and all faculties who provided assistance, encouragement, and feedback throughout the course of this project.Finally, we would like to thank our families for their unwavering support, understanding, and encouragement.

Abstract

Fault detection on process stations is crucial for ensuring operational efficiency and product quality in industrial settings .it compares the output we getting from an instrumentation system with the value that have been already stored and detect and shows the error in the system. In this project we covers various aspects including :Taking true value/theoretical value from MATPLOT LIBRARY ,reading values from sensor ,Building application for showing real time value . It highlights the importance of early fault detection, its impact on reducing downtime and maintenance costs, and enhancing overall productivity.. Moreover, it addresses the significance of real-time monitoring, anomaly detection, and predictive maintenance strategies in mitigating process station failures. Finally, it outlines future research directions and potential advancements in fault detection methodologies to address emerging industrial requirements and complexities

Contents

1. INTRODUCTION

Fault detection in process stations is essential for maintaining operational integrity and efficiency across industries. By leveraging advanced monitoring technologies and data analytics, these systems enable real-time detection and diagnosis of equipment abnormalities, minimizing downtime and enhancing productivity. In this overview, we explore the principles and applications of fault detection in process stations, highlighting its critical role in ensuring operational reliability, safety, and competitiveness. Why we need fault detectors globally? Fault detectors in process stations are essential globally for Operational Continuity, Risk Mitigation, Resource Optimization, reducing waste, Cost Reduction, Quality Assurance and Global Competitiveness

Statistics: Statistics on fault detection in process stations highlight its significance: 1. Downtime Costs 2. Maintenance Savings 3. Equipment Reliability 4. Environmental Impact 5. Safety 6. Industry Adoption. In summary, fault detection in process stations drives cost savings, improves reliability, enhances safety, and supports sustainability efforts. Existing solutions Here's a succinct overview of fault detection solutions in process stations, detailing the technology implemented, paper references, and the year of invention: 1. Manual Inspection: - Technology Implemented: Traditional visual checks and periodic maintenance. - Year of Invention: Since the advent of industrial processes. - References: Foundational practices in industrial maintenance. 2. Rule-Based Systems: - Technology Implemented: Predefined rules and thresholds for anomaly detection. - Year of Invention: 1960s. - References: - R.E. Kalman, "A New Approach to Linear Filtering and Prediction Problems" (1960). 3. Statistical Process Control (SPC): - Technology Implemented: Statistical methods like control charts. - Year of Invention: Early to mid-20th century. - References: - W.A. Shewhart, "Economic Control of Quality of Manufactured Product" (1931). 4. Machine Learning (ML): - Technology Implemented: Algorithms like neural networks for advanced analysis. - Year of Invention: 1950s onwards. - References: - G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function" (1989). 5. Data-Driven Approaches: - Technology Implemented: Leveraging large datasets for predictive maintenance. - Year of Invention: Early 2000s. - References: - H. Heimes, "The Evolution of Predictive Maintenance in the Automotive Industry" (2002). 6. IoT and Edge Computing: - Technology Implemented: Integration of IoT devices and edge computing. - Year of Invention: 2000s. - References: - T. Dohi and T. Osaki, ”Deterioration Modeling of Repairable Systems” (1993). 7. Artificial Intelligence (AI) and Deep Learning: - Technology Implemented: Advanced fault detection using AI and deep learning. - Year of Invention: 1980s onwards. - References: - Y. LeCun, Y. Bengio, and G. Hinton, ”Deep Learning” (2015). These solutions have evolved over time, with each iteration building upon the foundations laid by earlier technologies and methodologies

2. LITERATURE SURVEY

1 Abbas, A., Gani, R., Manan, Z. (2018). Fault Detection and Diagnosis in Industrial Systems: A Review. Processes, 6(10), 192: Covers statistical control, model-based, and emerging AI/ML techniques. Stresses the importance of diagnosis for minimizing downtime. Evaluates current trends and future research directions. Introduction to Industrial Systems: The paper would likely start by introducing industrial systems, which can range from manufacturing plants to power generation facilities, chemical processing plants, and more. These systems are complex and typically involve numerous interconnected components, machinery, and processes.

Importance of Fault Detection and Diagnosis: The paper would discuss why fault detection and diagnosis are crucial in industrial settings. Faults or abnormalities can lead to equipment breakdowns, production losses, safety hazards, and increased maintenance costs. Therefore, timely detection and diagnosis are essential to prevent or mitigate these negative consequences.

Fault Detection Techniques: The authors would likely review various techniques used for detecting faults in industrial systems. This could include methods such as statistical process control, machine learning algorithms, signal processing techniques, and physical modelling approaches.

2 Agrawal, R., Rangarajan, A. (2019). Fault detection, isolation, and diagnosis: A review. Annual Reviews in Control, 47, 290-307: Focuses on root cause identification to reduce diagnostic time. Accesses effectiveness of various algorithms. Discusses benefits and challenges of integrating ML in fault detection. Introduction to Fault Detection, Isolation, and Diagnosis (FDI): The paper would begin by introducing the concept of FDI in systems. FDI refers to the process of detecting the occurrence of faults or abnormalities, isolating the faulty component or subsystem, and diagnosing the root cause of the fault.

Importance and Applications: The authors would discuss the significance of FDI in various fields such as industrial automation, aerospace, automotive engineering, and process control. Efficient FDI systems are critical for ensuring the reliability, safety, and performance of complex systems.

Fault Detection Techniques: The paper would explore different methods for detecting faults in systems. This could include techniques such as statistical analysis, signal processing, machine learning algorithms, and sensor fusion approaches. These methods aim to identify deviations from normal system behaviour that may indicate the presence of faults.

Fault Isolation Methods: Once a fault is detected, the next step is to isolate the faulty component or subsystem. The authors would likely discuss strategies for narrowing down the possible causes of the fault based on available sensor data, system models, and diagnostic rules. This may involve techniques such as fault signature analysis, observer-based methods, and structural analysis.

Fault Diagnosis Approaches: After isolating the faulty component, the paper would cover methods for diagnosing the root cause of the fault. This could include model-based diagnosis, where the behaviour of the system is compared against a pre-existing model to identify discrepancies, or data-driven approaches that analyse historical data to infer the underlying fault mechanism.

- [3]Bartoszuk, K., Patan, K. (2017). Application of fault detection and diagnosis systems for industrial processes: A review. Control Engineering Practice, 60, 292-313 Introduction:

  Overview of the importance of fault detection and diagnosis in industrial processes. Introduces various methodologies and approaches used in FDD systems. Fault Detection Methods:

  Review of various fault detection methods including statistical methods, model-based methods, and hybrid methods are explained Discussed the advantages and disadvantages of each method. Fault Diagnosis Methods:

  Overview of fault diagnosis methods such as expert systems, knowledge-based systems, and artificial intelligence techniques. Discussion on the application of these methods in industrial processes. Integration of Fault Detection and Diagnosis Systems:

  Discussion on the integration of FDD systems into industrial processes. It also discussed the challenges and solutions related to integration.

3. SYSTEM DESCRIPTION

3.1 HARDWARE

3.1.1 ESP 32

The ESP32 is a powerful microcontroller and Wi-Fi/Bluetooth combo chip designed by Espressif Systems. It's widely used in IoT (Internet of Things) projects due to its versatility, low cost, and robust feature set. Key features include dual-core processors, Wi-Fi and Bluetooth connectivity, a variety of GPIO pins, analog-to-digital converters, and support for various communication protocols. Its capabilities make it ideal for a wide range of applications, from simple sensor networks to more complex projects requiring wireless connectivity and processing power. The

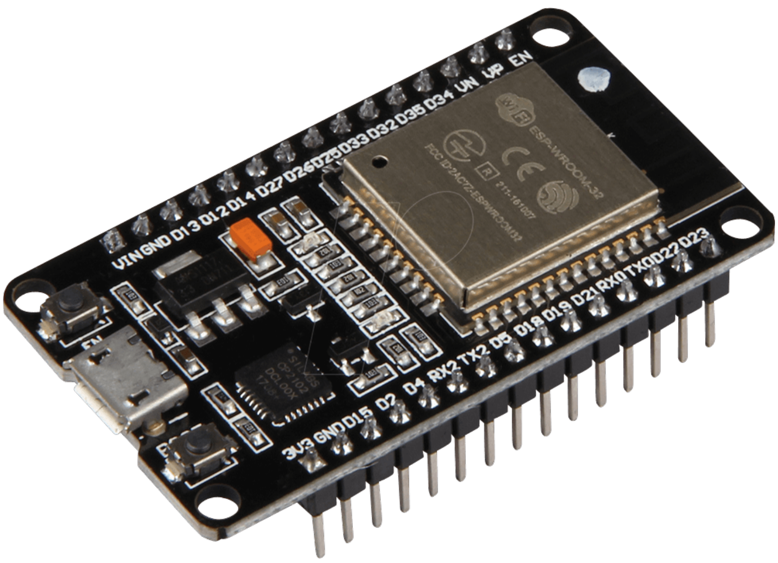


Figure 3.1: ESP 32

ESP32 microcontroller can be programmed using several programming languages and development environments, offering flexibility based on your preferences and project requirements.

Microcontroller	Tensilica L106
Operating Voltage	3.3V
Input Voltage (recommended)	2.5-3.6V
DC Current per I/O Pin	12 mA
Flash Memory	512 KB
SRAM	16 KB
EEPROM	4 KB

Table 3.1: Specifications Of ESP32

3.1.2 INA219 CURRENT SENSOR

The INA219 is a high-side current shunt and power monitor with an I2C interface, commonly used to measure current, voltage, and power in electronic systems. It accurately measures both the current passing through a shunt resistor and the voltage across that resistor, allowing for precise power calculations. With its high accuracy, wide input voltage range (0 to 26 volts), and low power consumption, the INA219 is suitable for various applications including battery management, power supply monitoring, energy monitoring, and motor control. Its configurability via the I2C interface, along with its ease of integration with microcontrollers and digital systems, makes it an ideal choice for applications requiring accurate current and power measurements.

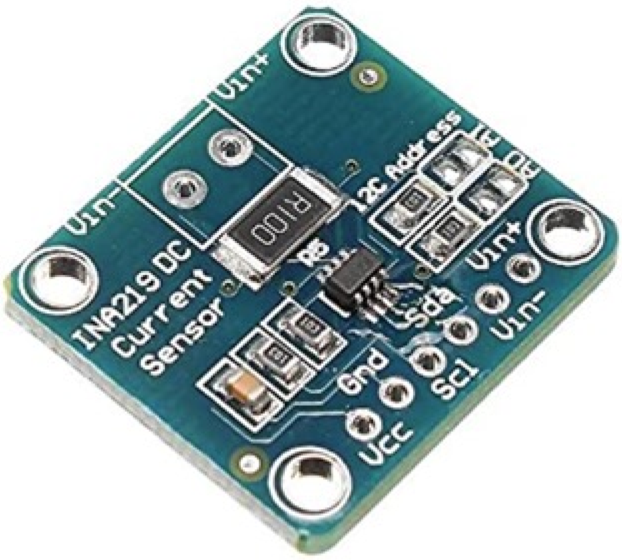


Figure 3.2: INA219 Current Sensor

The INA219 current sensor measures current by monitoring the voltage drop across an external shunt resistor. A low-ohmic shunt resistor is placed in series with the load whose current is to be measured. The INA219 then measures the voltage drop across this shunt resistor using an internal analog-to-digital converter (ADC). Using Ohm's Law (V = I * R), where V is the voltage across the shunt resistor, I is the current flowing through it, and R is the shunt resistor value, the INA219 calculates the current flowing through the load. This calculated current value is then available digitally via the I2C interface, along with other parameters such as voltage, power, and device configuration. This digital data can then be read by a microcontroller or digital system connected to the INA219, allowing for real-time monitoring and control of current, voltage, and power in the system.

Operating Voltage	3-5.5V DC
Common mode Voltage	26V DC
CMRR(dB)	100
Load Resistance (RL)	Depends on shunt resistor
Operating Temperature	$-40\,°C$ and $125\,°C$

Table 3.2: Specifications of MQ4 methane

3.2 SOFTWARE

3.2.1 ARDUINO IDE

Arduino IDE (Integrated Development Environment) is a software platform used for programming and developing projects with Arduino boards. Open-source electronics platform Arduino offers a straightforward and approachable way to develop interactive projects and prototypes.

The Arduino IDE is made to make writing code for Arduino boards simpler. It offers a code editor with tools like syntax highlighting, code completion, and error checking to make learning and writing Arduino sketches (programmes) simpler for newcomers. You can interface with the Arduino board and keep track of the data being sent or received using the built-in serial monitor that is part of the IDE. Its wide library support is one of the Arduino IDE's standout characteristics. It has a huge library of ready-to-use functions for a variety of activities, including controlling LEDs, reading sensors, interacting with other devices, and more. The development process is made simpler by these libraries, which encapsulate complicated operations into straightforward function calls. Both beginners and professional developers can use the Arduino IDE because it offers a condensed form of the C++ language.

The popular Arduino Uno, Arduino Mega, and Arduino Nano are just a few of the many Arduino boards that are supported by the Arduino IDE. Because it is compatible with Windows, macOS, and Linux operating systems, a large community of developers and enthusiasts can use it. Overall, Arduino IDE provides a user-friendly environment for programming Arduino boards, enabling individuals to bring their ideas to life through interactive and creative projects.

3.2.2 VISUAL STUDIO CODE

Visual Studio Code is a versatile and feature-rich code editor, suitable for a wide range of programming tasks, from simple scripting to large-scale application development

Visual Studio Code (VS Code) is a free source-code editor developed by Microsoft for Windows, Linux, and macOS. It offers a plethora of features including cross-platform support, extensive customization options, IntelliSense for smart code completions, an integrated terminal, built-in Git integration, debugging support, and a vast library of extensions. With its fast performance, productivity-boosting features, and strong community support, VS Code has become one of the most popular code editors among developers worldwide. It is widely used for various programming tasks including web development, cloud development, and programming in languages such as JavaScript, TypeScript, Python, Java, and C++.

Curve fitting of the values taken from the process station has been done in this platform. The curve fitted graph is given below

3.2.3 Android Studio Code

Android Studio is the official integrated development environment (IDE) for Android app development, provided by Google. It offers a comprehensive set of features including an intelligent code editor, visual layout editor, performance profiling tools, and built-in templates for common Android app components. With its built-in Android Emulator, developers can easily test their applications without the need for physical devices. Android Studio uses the Gradle build system for flexible and customizable builds and provides built-in support for version control systems like Git. It also offers full support for Kotlin, the official programming language for Android development, in addition to Java. Android Studio's extensive toolset, official support from Google, and strong community backing

The app built in this environment is made to show the error produced by the process stations.

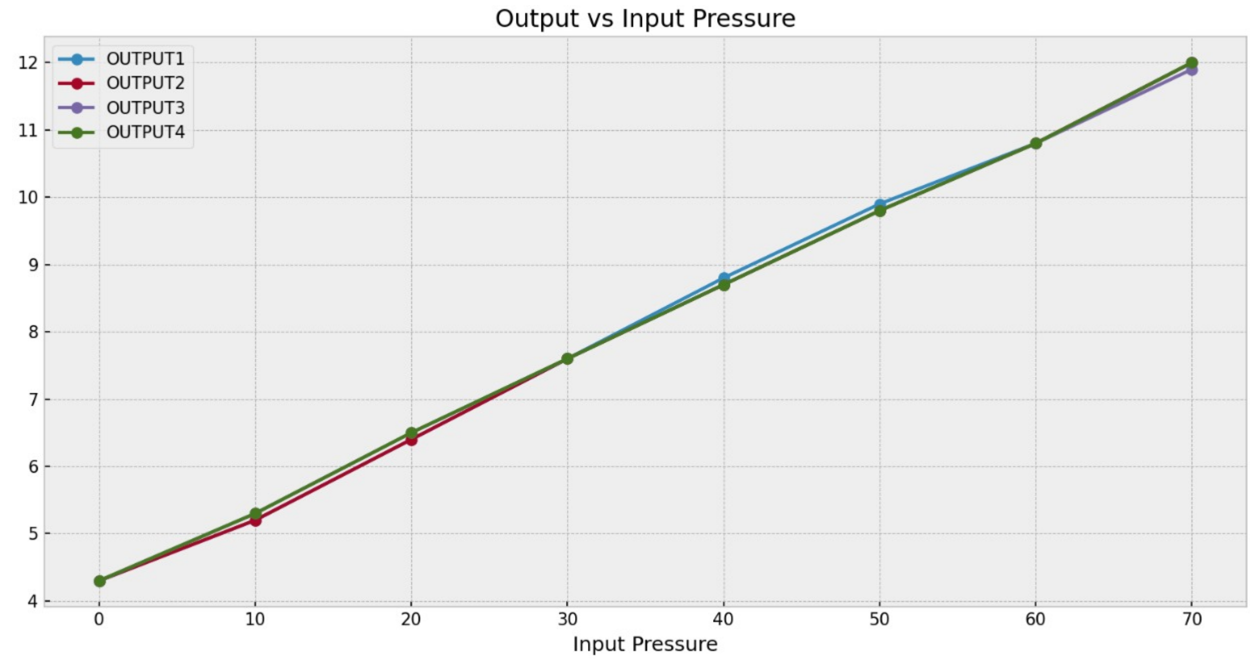


Figure 3.3: Curve fitted Graph

The measured value is also shown in the application

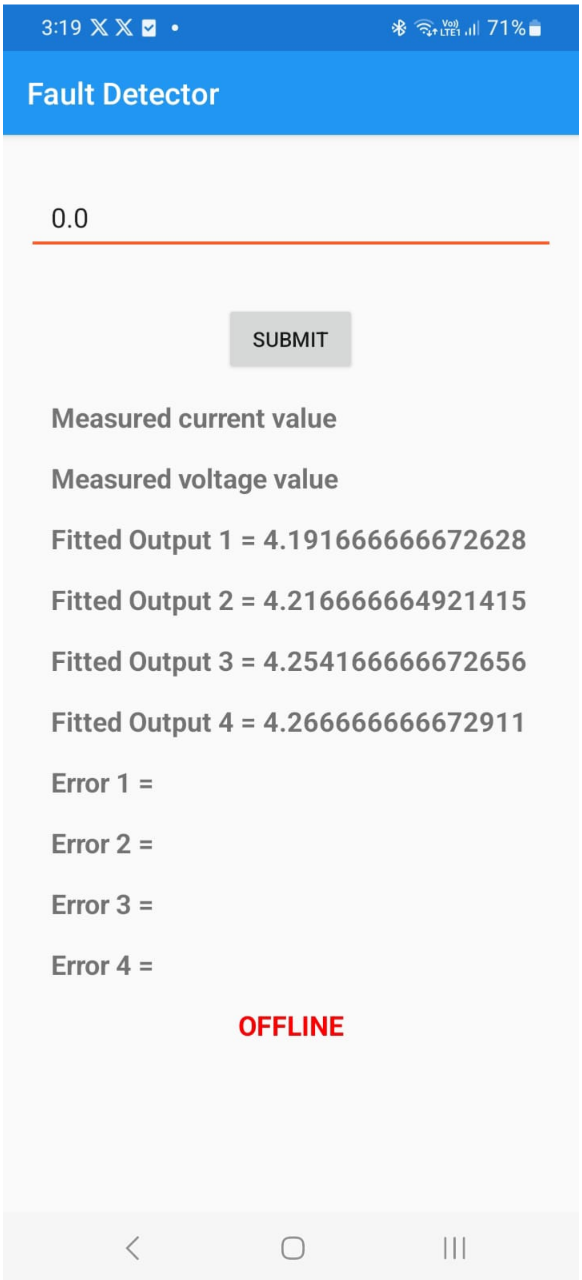


Figure 3.4: Fault Detector App

4. DESIGN AND ANALYSIS

4.1 METHODOLOGY

- Research : This project employs error diagnosis and data analysis techniques for fault detection in pressure process station
- Data Collection : Output of process station is tapped from various parts of the station with the help of a sensor
- Data Storage : Data is stored in a firebase and transmitted to the mobile application
- Fault Detection : The value of error and other parameters such as mean and accuracy of the process station is displayed in the mobile application

4.2 BLOCK DIAGRAM

The block diagram represents a system for monitoring electrical output from process stations using an INA219 current sensor, ESP8266 module, and a mobile app. The electrical output from process stations is measured by the INA219 current sensor. The ESP8266 module communicates with the INA219 sensor to receive current and voltage readings and sends this data to the mobile app via Wi-Fi. The mobile app displays the received data to the user and includes error handling functionality to show error messages if any abnormalities or faults are detected in the electrical processes being monitored. This system enables remote monitoring of electrical processes, providing real-time data and error notifications to the user via a mobile app, facilitating timely intervention and troubleshooting.

The INA219 current sensor accurately measures the current and voltage from the process stations, providing precise data for analysis. The ESP8266 module collects this data and establishes a Wi-Fi connection to transmit it to the mobile app. The mobile app, serving as the user

Figure 4.1: Enter Caption

interface, receives the data and provides real-time monitoring capabilities. Additionally, it employs error handling functionality to promptly notify the user of any detected abnormalities or faults.

4.3 WORKING

- Electrical Output from Process Stations:

  This represents the electrical output generated by various process stations. The output includes current and voltage measured from the output of controllers and transmitters of various process stations.

- INA219 Current Sensor:

  The INA219 current sensor is integrated into the system to accurately measure the current flowing through the electrical output from the process stations. Additionally, it can measure the voltage and power of the system. The electrical output from the controllers or the transmitters is read by this sensor

- ESP8266 Module:

  The ESP8266 module serves as the communication hub of the system. It interacts with the INA219 current sensor to collect current and voltage data. Subsequently, it establishes a Wi-Fi connection to transmit this data to the mobile app for further analysis and monitoring.

- Mobile App (Error Message Display):

  The mobile app acts as the user interface, providing real-time monitoring and control capabilities. It receives the current and voltage data from the ESP8266 module and displays it to the user in an understandable format.It shows the curve fitted values along with the error value and the measured value. Moreover, the app is equipped with error handling functionality. If any abnormalities or faults are detected in the electrical processes being monitored, the app generates and displays error messages to notify the user promptly.

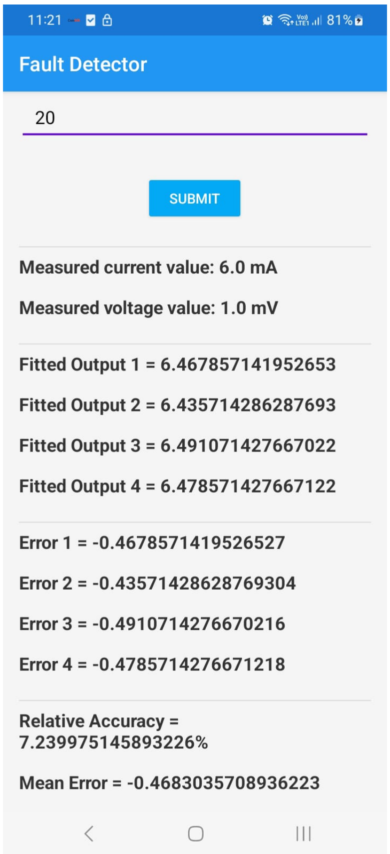
4.4 COST ANALYSIS

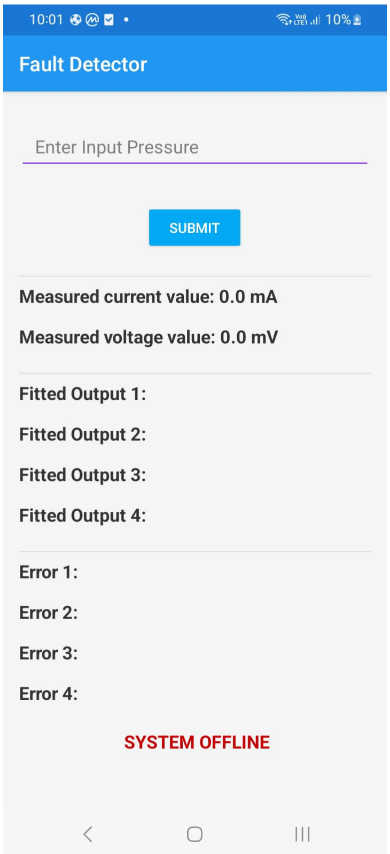
Item	Quantity	Unit Cost	Total Cost
ESP32	1	500	500
INA219 Sensor	1	400	400
Jumper Wires	-	-	20
Bread Board	1	60	60
Additional Expenses	-	-	41
Total Cost	-	-	1021

Table 4.1: Cost Analysis

5. RESULT

An app has been built and gained the knowledge on app development. we learned an insight on the curve fitting technique and its uses.





6. CONCLUSION

In summary, our ongoing project emphasizes the crucial need for efficient fault detection and management in process stations. Through our preliminary research and initial implementation efforts, we have identified key challenges and potential solutions in the realm of fault detection. As we continue to refine our methodologies and algorithms, we aim to achieve significant advancements in operational efficiency, safety, and productivity within process stations. Our work represents a promising step towards the development of robust fault management systems, with the ultimate goal of ensuring the reliability and optimization of industrial processes. We look forward to further progress and the eventual implementation of our solutions to address the pressing needs of the industry

7. REFERENCE

- We refered the books: ”fault detection and diagnosis in industrial system” by L.H. Chiang, E.L. Russell, and Braatz.

  ”Process systems Engineering: Volume 7: process operational safety and risk management” by Jose A. Romagnoli and Mabel critina cameiro fonseca.

- we also refered ”A review of data -driven fault detection and diagnosis methodologies for process system”by Venkatasubramanian.R.

- The paper works mentioned in the literature survey was mainly used as the references.

- We also mainly used the chatgpt(AI) ,the google(search engine),Bing(copilot) and You Tube for studying some of the procedures(such as downloading,learning etc) to do this project .

8. APPENDIX

8.1 Front end code

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp"
    android:background="#F5F5F5">

    <!-- Input Pressure EditText -->
    <EditText
        android:id="@+id/inputPressureEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:layout_marginBottom="16dp"
        android:hint="Enter Input Pressure"
        android:inputType="numberDecimal"
        android:minHeight="48dp"
        android:paddingStart="16dp"
        android:paddingEnd="16dp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:backgroundTint="#6200EE"
        android:textColor="#000000"
        android:textColorHint="#808080"/>
```

```xml

    <!-- Submit Button -->
    <Button
        android:id="@+id/submitButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/inputPressureEditText"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="16dp"
        android:text="Submit"
        android:backgroundTint="#03A9F4"
        android:textColor="#FFFFFF"
        android:paddingHorizontal="24dp"
        android:paddingVertical="12dp" />

    <!-- Separator for Real-Time Data -->
    <View
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:layout_below="@id/submitButton"
        android:layout_marginTop="24dp"
        android:background="#DDDDDD"/>

    <!-- Measured Current Value TextView -->
    <TextView
        android:id="@+id/measuredCurrentValueTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/submitButton"
        android:layout_marginTop="32dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Measured current value:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <!-- Measured Voltage Value TextView -->
```

```
    <TextView
        android:id="@+id/measuredVoltageValueTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/measuredCurrentValueTextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Measured voltage value:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <!-- Separator for Fitted Outputs -->
    <View
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:layout_below="@id/measuredVoltageValueTextView"
        android:layout_marginTop="24dp"
        android:background="#DDDDDD"/>

    <!-- Fitted Output TextViews -->
    <TextView
        android:id="@+id/fittedOutput1TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/measuredVoltageValueTextView"
        android:layout_marginTop="32dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Fitted Output 1:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <TextView
        android:id="@+id/fittedOutput2TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_below="@id/fittedOutput1TextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Fitted Output 2:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <TextView
        android:id="@+id/fittedOutput3TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/fittedOutput2TextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Fitted Output 3:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <TextView
        android:id="@+id/fittedOutput4TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/fittedOutput3TextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Fitted Output 4:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <!-- Separator for Errors -->
    <View
        android:layout_width="match_parent"
        android:layout_height="1dp"
        android:layout_below="@id/fittedOutput4TextView"
```

```
        android:layout_marginTop="24dp"
        android:background="#DDDDDD"/>

    <!-- Error TextViews -->
    <TextView
        android:id="@+id/error1TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/fittedOutput4TextView"
        android:layout_marginTop="32dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Error 1:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <TextView
        android:id="@+id/error2TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/error1TextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Error 2:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <TextView
        android:id="@+id/error3TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/error2TextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Error 3:"
```

```
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <TextView
        android:id="@+id/error4TextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/error3TextView"
        android:layout_marginTop="16dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:text="Error 4:"
        android:layout_alignParentStart="true"
        android:textColor="#333333" />

    <!-- Status TextView -->
    <TextView
        android:id="@+id/statusTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/error4TextView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="24dp"
        android:textSize="18sp"
        android:textStyle="bold"
        android:textColor="#FF0000"
        android:text="OFFLINE" />

</RelativeLayout>
```

Listing 8.1: Front end code

8.2 Backend code

```
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.opencsv.CSVReader;
import com.opencsv.exceptions.CsvValidationException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MainActivity";


    private DatabaseReference readingsRef;


    private EditText inputPressureEditText;
    private Button submitButton;
```

```
    private TextView measuredCurrentValueTextView;
    private TextView measuredVoltageValueTextView;
    private TextView fittedOutput1TextView;
    private TextView fittedOutput2TextView;
    private TextView fittedOutput3TextView;
    private TextView fittedOutput4TextView;
    private TextView error1TextView;
    private TextView error2TextView;
    private TextView error3TextView;
    private TextView error4TextView;
    private TextView statusTextView;


    private List<CsvData> csvDataList = new ArrayList<>();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.new_layout);


    readingsRef=
    FirebaseDatabase.getInstance().getReference().child("readings");


        inputPressureEditText =
           findViewById(R.id.inputPressureEditText);
        submitButton = findViewById(R.id.submitButton);
        measuredCurrentValueTextView =
           findViewById(R.id.measuredCurrentValueTextView);
        measuredVoltageValueTextView =
           findViewById(R.id.measuredVoltageValueTextView);
        fittedOutput1TextView =
           findViewById(R.id.fittedOutput1TextView);
        fittedOutput2TextView =
           findViewById(R.id.fittedOutput2TextView);
```

```
        fittedOutput3TextView =
           findViewById(R.id.fittedOutput3TextView);
        fittedOutput4TextView =
           findViewById(R.id.fittedOutput4TextView);
        error1TextView = findViewById(R.id.error1TextView);
        error2TextView = findViewById(R.id.error2TextView);
        error3TextView = findViewById(R.id.error3TextView);
        error4TextView = findViewById(R.id.error4TextView);
        statusTextView = findViewById(R.id.statusTextView);


        readCsvFile();


        submitButton.setOnClickListener(new
           View.OnClickListener() {
            @Override
            public void onClick(View v) {
                handleButtonClick();
            }
        });

        Realtime Database
        readingsRef.addValueEventListener(new
           ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot
               dataSnapshot) {
                if (dataSnapshot.exists()) {

    double current =
       dataSnapshot.child("current").getValue(Double.class);
    double voltage =
       dataSnapshot.child("voltage").getValue(Double.class);


```

```
    measuredCurrentValueTextView.setText("Measured current value: " + current + " mA");
    measuredVoltageValueTextView.setText("Measured voltage value: " + voltage + " mV");


                    updateStatus(current);


                    calculateAndDisplayErrors(current);
                } else {
                    Log.e(TAG, "No data available");
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                Log.e(TAG, "Firebase Database error: " + databaseError.getMessage());
            }
        });
    }


    private void readCsvFile() {
        try (InputStream inputStream = getResources().openRawResource(R.raw.fitted_outputs);
             BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream));
             CSVReader csvReader = new CSVReader(reader)) {


            csvReader.readNext();

            String[] nextRecord;
```

```
                while ((nextRecord = csvReader.readNext()) !=
                    null) {
                    CsvData csvData = new CsvData();
csvData.setInputPressure(Double.parseDouble(nextRecord[0]));
csvData.setFittedOutput1(Double.parseDouble(nextRecord[1]));
csvData.setFittedOutput2(Double.parseDouble(nextRecord[2]));
csvData.setFittedOutput3(Double.parseDouble(nextRecord[3]));
csvData.setFittedOutput4(Double.parseDouble(nextRecord[4]));
                    csvDataList.add(csvData);
                }
            } catch (IOException | CsvValidationException e) {
                Log.e(TAG, "Error reading CSV file", e);
            }
      }


    private void handleButtonClick() {

        String inputPressureStr =
            inputPressureEditText.getText().toString().trim();
        if (inputPressureStr.isEmpty()) {
            Log.e(TAG, "Input pressure is empty");
            return;
        }

        try {
            double inputPressure =
                Double.parseDouble(inputPressureStr);


            CsvData closestMatch =
                findClosestMatch(csvDataList, inputPressure);
            if (closestMatch != null) {

                fittedOutput1TextView.setText("Fitted Output
                    1 = " + closestMatch.getFittedOutput1());
```

```
                fittedOutput2TextView.setText("Fitted Output
                   2 = " + closestMatch.getFittedOutput2());
                fittedOutput3TextView.setText("Fitted Output
                   3 = " + closestMatch.getFittedOutput3());
                fittedOutput4TextView.setText("Fitted Output
                   4 = " + closestMatch.getFittedOutput4());


    double
       measuredCurrent=Double.parseDouble(measuredCurrentValueTextView.ge
                          .replace("Measured current value: ",
                             "").replace(" mA", ""));
                calculateAndDisplayErrors(measuredCurrent);
            } else {
                Log.e(TAG, "No matching data found for input
                   pressure: " + inputPressure);
            }
        } catch (NumberFormatException e) {
            Log.e(TAG, "Error parsing input pressure: " +
               inputPressureStr, e);
        }
    }


    private void calculateAndDisplayErrors(double
       measuredCurrent) {
        if (csvDataList.isEmpty()) {
            Log.e(TAG, "CSV data list is empty");
            return;
        }

        try {
double fittedOutput1 =
   Double.parseDouble(fittedOutput1TextView.getText().toString()
                      .replace("Fitted Output 1 = ",
                         "").trim());
```

```
 double fittedOutput2 =
    Double.parseDouble(fittedOutput2TextView.getText().toString()
                    .replace("Fitted Output 2 = ",
                        "").trim());
double fittedOutput3=
   Double.parseDouble(fittedOutput3TextView.getText().toString()
                    .replace("Fitted Output 3 = ",
                        "").trim());
double fittedOutput4 =
   Double.parseDouble(fittedOutput4TextView.getText().toString()
                    .replace("Fitted Output 4 = ",
                        "").trim());

            double error1 = measuredCurrent - fittedOutput1;
            double error2 = measuredCurrent - fittedOutput2;
            double error3 = measuredCurrent - fittedOutput3;
            double error4 = measuredCurrent - fittedOutput4;

            error1TextView.setText("Error 1 = " + error1);
            error2TextView.setText("Error 2 = " + error2);
            error3TextView.setText("Error 3 = " + error3);
            error4TextView.setText("Error 4 = " + error4);
        } catch (NumberFormatException e) {
            Log.e(TAG, "Error parsing fitted output values",
               e);
        }
    }


    private CsvData findClosestMatch(List<CsvData> data,
       double inputPressure) {
        CsvData closestMatch = null;
        double minDifference = Double.MAX_VALUE;
        for (CsvData csvData : data) {
            double difference =
               Math.abs(csvData.getInputPressure() -
               inputPressure);
```

```
            if (difference < minDifference) {
                minDifference = difference;
                closestMatch = csvData;
            }
        }
        return closestMatch;
    }


    private void updateStatus(double measuredCurrent) {
        if (measuredCurrent < 4.0) {
statusTextView.setText(" SYSTEM OFFLINE");
statusTextView.setTextColor(getResources().getColor(android.R.color.holo_
        } else {
statusTextView.setText(" SYSTEM ONLINE");
statusTextView.setTextColor(getResources().getColor(android.R.color.holo_
        }
    }


    private static class CsvData {
        private double inputPressure;
        private double fittedOutput1;
        private double fittedOutput2;
        private double fittedOutput3;
        private double fittedOutput4;


        public double getInputPressure() {
            return inputPressure;
        }

        public void setInputPressure(double inputPressure) {
            this.inputPressure = inputPressure;
        }

        public double getFittedOutput1() {
```

```
                return fittedOutput1;
            }

            public void setFittedOutput1(double fittedOutput1) {
                this.fittedOutput1 = fittedOutput1;
            }

            public double getFittedOutput2() {
                return fittedOutput2;
            }

            public void setFittedOutput2(double fittedOutput2) {
                this.fittedOutput2 = fittedOutput2;
            }

            public double getFittedOutput3() {
                return fittedOutput3;
            }

            public void setFittedOutput3(double fittedOutput3) {
                this.fittedOutput3 = fittedOutput3;
            }

            public double getFittedOutput4() {
                return fittedOutput4;
            }

            public void setFittedOutput4(double fittedOutput4) {
                this.fittedOutput4 = fittedOutput4;
            }
        }
}
```

Listing 8.2: Backend code

8.3 Curve Fitting Code

```
import pandas as pd
import numpy as np
from scipy.optimize import curve_fit

# Generate input pressure values from 0 to 70 with a division
   of 0.1
x_new = np.arange(0, 70.1, 0.1)

# Define the original data
df = pd.read_csv('BOOK1.csv')

# Extract original input pressure and outputs
x_original = df['INPUT PRESSURE']
y1 = df['OUTPUT1']
y2 = df['OUTPUT2']
y3 = df['OUTPUT3']
y4 = df['OUTPUT4']

# Define a quadratic model function
def quadratic_model(x, a, b, c):
    return a * x**2 + b * x + c

# Perform curve fitting for each output using the quadratic
   model and new input pressure values
popt1, pcov1 = curve_fit(quadratic_model, x_original, y1)
popt2, pcov2 = curve_fit(quadratic_model, x_original, y2)
popt3, pcov3 = curve_fit(quadratic_model, x_original, y3)
popt4, pcov4 = curve_fit(quadratic_model, x_original, y4)

# Generate fitted curves using the quadratic model and new
   input pressure values
y1_fit = quadratic_model(x_new, *popt1)
y2_fit = quadratic_model(x_new, *popt2)
y3_fit = quadratic_model(x_new, *popt3)
y4_fit = quadratic_model(x_new, *popt4)
```

```

# Create DataFrame for fitted data
fitted_data = pd.DataFrame({
    'Input Pressure': x_new,
    'Fitted Output1': y1_fit,
    'Fitted Output2': y2_fit,
    'Fitted Output3': y3_fit,
    'Fitted Output4': y4_fit
})

# Save fitted data to CSV
fitted_data.to_csv('fitted_data_progression.csv', index=False)
```

Listing 8.3: Curve Fitting Code